WEST Search History

Hide Items | Restore | Clear | Cancel

DATE: Friday, June 03, 2005

Hide?	Set Name	Query	Hit Count
		DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=ADJ	
☐	L22	L19 and l9	0
☐	L21	L19 and l7	0
☐	L20	L19 and l12	0
☐	L19	19990630	2
☐	L18	(asynchronous near3 (RPC or (procedure adj call))).ti.	3
☐	L17	19990630	9
☐	L16	l12 and (RPC same asynchronous)	1
☐	L15	l12 and RPC	37
☐	L14	l12 and (RPC near8 asynchronous)	1
☐	L13	l12 and (RPC near asynchronous)	0
☐	L12	(compare or match or comparing or matching) near8 (parameter or ID or identification) near8 (store or stored)	11360
☐	L11	19990630	4
☐	L10	L9 and l8	6
☐	L9	(store or storing) near8 (identification or ID or parameter)	91129
☐	L8	L7 and RPC	41
☐	L7	(match or matching) near8 request near8 response	1323
☐	L6	19990630	4
☐	L5	L4 and (match or matching or synchronize or synchronizing or compare or comparing)	12
☐	L4	L3 and (memory or cache)	12
☐	L3	RPC same asynchronous same (identify or identification or ID)	13
☐	L2	RPC near8 asynchronous near8 (identify or identification or ID)	2
☐	L1	RPC near asynchronous near8 (identify or identification or ID)	0

END OF SEARCH HISTORY

Previous Doc Next Doc Go to Doc#
First Hit Fwd Refs

Generate Collection

L17: Entry 5 of 9 File: USPT Sep 1, 1998

DOCUMENT-IDENTIFIER: US 5802298 A
TITLE: Defect-free type remote procedure call system and method thereof

Abstract Text (1):
When a client sends an RPC request that requests a server for a service, the client adds identification information to the RPC request. When an active server receives an RPC request, it stores the identification information thereof in a stable area that is not destroyed even if a defect takes place in the client or the server, and executes the requested service. When a defect takes place in the active server, the backup server takes over the process of the active server. A PALIB of the backup server compares the identification information of the RPC request re-sent from the client with the identification information in the stable area. When they match, the PALIB determines that the RPC request is redundant. The backup server performs a redundant process and sends back the correct result to the client.

Application Filing Date (1):
19960826

Brief Summary Text (10):
In a conventional defect-free RPC (remote procedure call) system, to accomplish a highly available computer system having a plurality of processes, an automatic re-sending system or a manual re-sending system for a message is used. Thus, even if a defect takes place in a sender process, the sender process are not required to countermeasures thereof. With the automatic/manual re-sending system, even if a defect takes place in a fault-tolerant (FT) type processing system while it is executing an RPC, the service can be continued according to the RPC request that has not been sent upon occurrence of the defect is re-sent.

Brief Summary Text (13):
Now, assume that a service request is sent from the processing portion A, the service request is relayed to the processing portion B, the service request is received by the processing portion C, and then a particular process is performed by the processing portion C. After the service request is sent to the processing portion C, when a defect takes place in the relay processing portion B, a substitute processing portion B' of the processing portion B is prepared. Thereafter, the processing portion A reissues a service request with a re-sending function of the communication system. The substitute processing portion B' relays the re-sent service request and the processing system C resumes the service. A processing portion that sends a service request, as in the processing portion A, is referred to as a client. A processing system that provides a service, as in the processing portion C, is referred to as a server. Further, a processing portion that relays a service, as in the processing portions B and B', is referred to as a relay server. When the server executes a service corresponding to an RPC service request received from a client, after the server has performed the service, the server may not return to the state it was in before it performed the service. In this case, the RPC is referred to as an non-idempotent RPC. In the case of the non-idempotent RPC, even if the client re-sends the same message and the server re-executes the service request, the same result may not be obtained or an improper side effect may take place.

Brief Summary Text (14):
An example of the non-idempotent process is a deletion of a file or a directory. Now, assume the following situation. A client sends a first delete request to a server. The server executes this request. While the result is being returned to the client, a defect takes place. At this point, if the delete request is re-sent and the request is simply re-executed, the server determines that the client has requested a deletion of a file or a directory that does not exist. Thus, the server returns an error message to the client. In the RPC re-sending operation that requests such a non-idempotent process, a re-sending operation that has been executed by the server and that redundantly takes place is referred to as a redundant re-sending operation.

Brief Summary Text (15):
In a server-client type processing system, when the server is performing a non-idempotent RPC process, if a particular transaction is re-sent due to a particular cause, unless the server detects the transaction is resent and returns the lost process result to the client, it cannot be said that the server returns a correct result to the client. A system that detects the re-sent request and returns a correct result upon occurrence of the re-sent message is a redundantly sent message detecting system. The redundantly sent message detecting system can be accomplished by various methods. As a simple method, a processing portion that detects a redundantly sent message has stored a transaction of a non-idempotent process and the result thereof. When the same transaction is requested, the processing portion returns the stored result.

Brief Summary Text (16):
However, the conventional defect-free type RPC system has the following problems.

Brief Summary Text (22):
An object of the present invention is to provide a defect-free type RPC system for easily re-sending a request message and correctly returning a reply message, in the case that a partial defect takes place in a highly available computer system and a method thereof.

Brief Summary Text (23):
The present invention relates to a defect-free type RPC system in which a first process requests a second process for a service through a communication network, and the second process returns the result of the request to the first process. The defect-free type communication system has a port managing portion, an identifier adding portion, a sending portion, a detecting portion, a replying portion, a first storing portion, a second storing portion, and a relaying portion.

Detailed Description Text (10):
When the identification information of the second request message received after the first request message matches the identification information stored in the redundantly sent message process table, the detecting portion 4 determines that the second request message is a redundantly re-sent message.

Detailed Description Text (21):
FIG. 2 shows a defect managing system for use with a defect-free type RPC system according to the present invention. In FIG. 2, a port to a set of processes that provide services to a client 14 (request sender process) is referred to as a port alias. In this example, a port alias 15 is connected to a port of a primary server 16 and a port of a backup server 17.

Detailed Description Text (29):
Thus, a message that is sent from a process that requests a service to a port alias automatically reaches a process that provides the service. By inquiring a port alias instead of a port that provides a service, with the same identifier, even

after a defect takes place, the communication can be continued with a process that provides the service. The port alias is a virtual address that is used when a RPC is requested to a process that is providing a service. The port alias does not vary even if a process is taken over by a substitute system due to a defect of the active system.

Detailed Description Text (30):
Likewise, the port alias can be defined for a process that requests a service. In this case, the port alias represents a port of a set of a plurality of request sender processes. The port alias is a virtual sender that is used when an RPC is requested. The virtual sender is also managed by the PAM 12 corresponding to the port alias table 13. When a service is taken over by a backup system, the port alias is not varied.

Detailed Description Text (38):
FIG. 5 shows a redundantly sent message detecting system of a defect-free type RPC supporting system. In FIG. 5, a client 21 is composed of an SSC (server specific code) 22 for describing a main routine and a PALIB 23 that is a library called by the SSC 22. The client 21 has a port 24 and a port alias 25 in which the port 24 is included. A non-idempotent server 31 has an active server and a backup server that has the same function of the active server so as to accomplish the defect-free characteristic. In addition, the non-idempotent server 31 has a port alias 39 that is a common port of the active server and the backup server. The active server and the backup server share a stable area (SA) 35 that is a storage region that is not destroyed when a defect takes place in these servers so as to take over information each other. When a defect takes place in the active server, the backup server obtains information from the SA 35 so as to match the information of the active server and the information of the backup server.

Detailed Description Text (41):
When the client 21 sends an RPC message to the non-idempotent server 31, the PALIB 23 that is the communication system of the client 21 adds unique identifiers (ID) of RPC requests to all RPC messages and sends the resultant messages. As unique IDs, a combination of {srcAlias, srcPort, transId} or a combination of {srcAlias, incarnation, transId} is used. An ID added to an RPC message is stored in the SA 35 of the server 31 and used to identify a received message. srcAlias is an identifier that identifies a port alias of a sender. The srcAlias of a first sent message is the same as the srcAlias of the re-sent message. srcPort is an identifier of a port of a sender process of the first sending operation. incarnation is an incarnation number of an active process of the first sending operation. The incarnation number is a generation number that varies whenever the process of the active system is taken over by the process of the backup system. The incarnation number is managed by the FTM 11. The transId is a unique ID of an FT transaction. The transId of the first sending operation is the same as the transID of the re-sending operation. transId may be unique in each site, in each port alias, or in each process.

Detailed Description Text (42):
The user of the server 31 of the defect-free type RPC supporting system designates a desired operation to the port alias 39 when he or she receives an RPC redundant message that is being currently processed. For example, the user designates to the port alias 39 the content of a message that the port alias 39 automatically sends back to the user. The port alias 39 stores the message and uses it in a redundantly sent message detecting process (that will be described later). In addition, the user can cause the port alias 39 to call a redundantly sent message handler that is a function of a redundantly sent message process. At this point, the port alias 39 stores a startup parameter that is given whenever the redundantly sent message handler is called. The port alias 39 calls the redundantly sent message handler in the redundantly sent message detecting process corresponding to the parameter designated by the user.

Detailed Description Text (43):
Next, the redundantly sent message detecting process will be described. In an non-idempotent process, the active server notifies the PALIB 33 that a redundant process may be performed before a roll-back disable state takes place while the RPC is being processed. This operation is referred to as a notification of a roll forward point. The roll forward point is a specific point in the main routine of the SSC 32. When the process is performed exceeding the roll forward point, the former state cannot be restored. According to this notification, the PALIB 33 registers information representing that the roll forward point has been notified and information for detecting the redundantly sent message (ID of the message and so forth) to redundantly sent message process tables in the SA 35 or in other memory regions managed by the PALIB 33 and 37.

Detailed Description Text (44):
When a defect takes place in the active server before the notification of the roll forward point, since the process of the RPC performed in the active server has not been in the roll back disable state, a re-sent message performed by the re-sending system is not treated as a redundantly sent message. Thus, in a new active server, by simply re-executing a RPC request, the process is correctly continued. However, after the roll forward point has been notified, until the reply of the RPC is returned to the client 21, if a defect takes place in the client 21 or the server 31, the message resent by the re-sending system becomes a redundantly sent message.

Detailed Description Text (45):
The SSC 32 causes the PALIB 33 to perform a redundantly sent message process when the content of the process that is performed upon detection of a redundantly sent message is decided. The PALIBs 33 and 37 linked to processes of the active server and the backup server each compare the received RPC message with the information for detecting a redundantly sent message in the redundantly sent message process table, and check whether or not the roll forward point has been notified so as to determine whether or not the received message is a redundantly sent message. When a redundantly sent message is detected, a redundantly sent message process registered to the message by the process is activated.

Detailed Description Text (49):
The second example is a redundantly sent message handler (redundant message handler) process. In this case, a redundantly sent message handler, that is a function called upon detection of a redundantly sent message, is designated for each message. When the server 31 detects a redundantly sent message, it calls a registered redundantly sent message handler. The SSC 32 registers a redundant handler calling process through the PALIB 33 when the process for the redundantly sent message is registered. The PALIB 33 changes the item of the related message of the redundantly sent message process table from the roll forward point notification completion state to the redundant handler registration completion state and registers an argument to be passed to the redundantly sent message handler. When the redundantly sent message is detected, the registered redundantly sent message handler is called with the argument. The called redundantly sent message handler re-calculates the result of the RPC process that has already been executed and securely sends back the reply to the client 21.

Detailed Description Text (51):
FIG. 6 shows an example of an automatic re-sending operation of an RPC request and a redundantly sent message detection in the case that an active server shown in FIG. 5 exceeds a roll forward point and a defect takes place therein. In FIG. 6, a PALIB 23 of a client 21 adds an ID {srcAlias, srcPort, transId} to a message and sends the resultant message to a port alias 39 of a server 31. srcAlias and srcPort represent a port alias 25 and an identifier of a port 24, respectively. At this point, the active server registers the ID of the received message to a redundantly sent message process table in an SA 35 or in another memory region managed by the

PALIB 23.

Detailed Description Text (52):
When the redundantly sent message process is an automatic result replying process, the defect-free type RPC supporting system performs the following operation.

Detailed Description Text (57):
When the redundantly sent message process is a redundantly sent message handler process, the defect-free type RPC supporting system performs the following operation.

Detailed Description Text (64):
FIG. 7 shows an example of manual re-sending operation of RPC request and a redundantly sent message detection in the case that a saver exceeds a roll forward point and a detect take place a client.

Detailed Description Text (67):
When the identifier (srcPort) of the sender port is sent along with a message, the defect-free type RPC supporting system performs the following operation.

Detailed Description Text (71):
When an incarnation number of a message is sent along with the message, the defect-free type RPC supporting system performs the following operation.

Detailed Description Text (76):
FIG. 8 is a flow chart showing a process of a PALIB called from a main routine when a client sends a request message. In FIG. 8, when the process is started, the PALIB adds an identifier to a request message (at step S1) and sends the request message to the server corresponding to an RPC request function (at step S2). As a result, the process is completed.

Detailed Description Text (132):
According to the present invention, after an RPC request is issued in a highly available computer system, when a defect takes place in hardware or software, a request message can be easily re-sent and a correct reply message can be sent back. In particular, with a server that performs an non-idempotent process, a message redundantly sent from the client can be handled. In addition, in the case that a relay processing system is disposed in a distribution system, when a defect takes place therein, a normal operation state can be restored.

Previous Doc Next Doc Go to Doc#